<https://leetcode.com/problems/pancake-sorting/description/>

Given an array of integers arr, sort the array by performing a series of **pancake flips**.

In one pancake flip we do the following steps:

1. Choose an integer k where 1 <= k <= arr.length.
2. Reverse the sub-array arr[0...k-1] (0-indexed).

For example, if arr = [3,2,1,4] and we performed a pancake flip choosing k = 3, we reverse the sub-array [3,2,1], so arr = [1,2,3,4] after the pancake flip at k = 3.

Return **an array of the**k**-values corresponding to a sequence of pancake flips that sort**arr. Any valid answer that sorts the array within 10 \* arr.length flips will be judged as correct.

**Example 1:**

**Input:** arr = [3,2,4,1]

**Output:** [4,2,4,3]

**Explanation:**

We perform 4 pancake flips, with k values 4, 2, 4, and 3.

Starting state: arr = [3, 2, 4, 1]

After 1st flip (k = 4): arr = [1, 4, 2, 3]

After 2nd flip (k = 2): arr = [4, 1, 2, 3]

After 3rd flip (k = 4): arr = [3, 2, 1, 4]

After 4th flip (k = 3): arr = [1, 2, 3, 4], which is sorted.

**Example 2:**

**Input:** arr = [1,2,3]

**Output:** []

**Explanation:** The input is already sorted, so there is no need to flip anything.

Note that other answers, such as [3, 3], would also be accepted.

**Attempt 1: 2024-01-05**

**Solution 1: Sorting + Two Pointers (60min)**

**本题的做法非常像Bubble Sort**

class Solution {

    public List<Integer> pancakeSort(int[] arr) {

        List<Integer> result = new ArrayList<>();

        // The count of already sorted largest elements piling up

        // at rightmost indexes, the intention is using it as

        // blocker of infinite for loop since every time we

        // restore the index 'i' back to -1 prepare for next

        // for loop, if no additional blocker, the for loop

        // won't end

        int sorted\_count = 0;

        int len = arr.length;

        int target = arr.length;

        int largestIndex = -1;

        int lastIndex = len - 1;

        // Definition: All integers in arr are unique

        // (i.e. arr is a permutation of the integers from 1 to arr.length).

        for(int i = 0; i < len; i++) {

            if(arr[i] == target && sorted\_count < len) {

                largestIndex = i;

                // Next round 'target' will be 1 less than current round

                target--;

                // First pancake flip to switch the largest element to

                // the first position(which always 0)

                // The condition 'largestIndex != 0' as no need reverse in place

                if(largestIndex != 0) {

                    reverse(arr, 0, largestIndex);

                    // The output is 1-based

                    result.add(largestIndex + 1);

                }

                // Second pancake flip to reverse the whole unsorted array

                // (the sorted part are built by largest elements sorted in

                // ascending order piling up at rightmost indexes), to switch

                // the largest element to the last position(which gradually

                // move from len - 1 to 0)

                // The condition 'lastIndex != 0' as no need reverse in place

                if(lastIndex != 0) {

                    reverse(arr, 0, lastIndex);

                    // The output is 1-based

                    result.add(lastIndex + 1);

                }

                lastIndex--;

                sorted\_count++;

                // Restore 'i' prepare for next for loop

                i = -1;

            }

        }

        return result;

    }

    private void reverse(int[] arr, int i, int j) {

        while(i < j) {

            int tmp = arr[i];

            arr[i] = arr[j];

            arr[j] = tmp;

            i++;

            j--;

        }

    }

}

Time Complexity: O(N^2), because every time we restore 'i' back to 0, it cause each iteration actually start a fresh, so one for loop but N^2

Space Complexity: O(N)

**Refer to**

<https://grandyang.com/leetcode/969/>

这道题给了长度为n的数组，由1到n的组成，顺序是打乱的。现在说我们可以任意翻转前k个数字，k的范围是1到n，问怎么个翻转法能将数组翻成有序的。题目说并不限定具体的翻法，只要在 10\*n 的次数内翻成有序的都是可以的，任你随意翻，就算有无效的步骤也无所谓。题目中给的例子1其实挺迷惑的，因为并不知道为啥要那样翻，也没有一个固定的翻法，所以可能会误导大家。必须要自己想出一个固定的翻法，这样才能应对所有的情况。博主想出的方法是每次先将数组中最大数字找出来，然后将最大数字翻转到首位置，然后翻转整个数组，这样最大数字就跑到最后去了。然后将最后面的最大数字去掉，这样又重现一样的情况，重复同样的步骤，直到数组只剩一个数字1为止，在过程中就把每次要翻转的位置都记录到结果 res 中就可以了，注意这里 C++ 的翻转函数 reverse 的结束位置是开区间，很容易出错，参见代码如下：

class Solution {

public:

vector<int> pancakeSort(vector<int>& arr) {

vector<int> res;

while (arr.size() > 1) {

int n = arr.size(), i = 0;

for (; i < n; ++i) {

if (arr[i] == n) break;

}

res.push\_back(i + 1);

reverse(arr.begin(), arr.begin() + i + 1);

res.push\_back(n);

reverse(arr.begin(), arr.end());

arr.pop\_back();

}

return res;

}

};

上面的方法可以略微优化一下，并不用真的从数组中移除数字，只要确定个范围就行了，右边界不断的缩小，效果跟移除数字一样的

class Solution {

public:

vector<int> pancakeSort(vector<int>& arr) {

vector<int> res;

for (int i = arr.size(), j; i > 0; --i) {

for (j = 0; arr[j] != i; ++j);

reverse(arr.begin(), arr.begin() + j + 1);

res.push\_back(j + 1);

reverse(arr.begin(), arr.begin() + i);

res.push\_back(i);

}

return res;

}

};

**Java flip the largest number to the tail**

**Refer to**

<https://leetcode.com/problems/pancake-sorting/solutions/214200/java-flip-the-largest-number-to-the-tail/>

1. Find the largest number
2. Flip twice to the tail

Time: O(N^2)

Flips: 2\*N

class Solution {

public List<Integer> pancakeSort(int[] A) {

List<Integer> result = new ArrayList<>();

int n = A.length, largest = n;

for (int i = 0; i < n; i++) {

int index = find(A, largest);

flip(A, index);

flip(A, largest - 1);

result.add(index + 1);

result.add(largest--);

}

return result;

}

private int find(int[] A, int target) {

for (int i = 0; i < A.length; i++) {

if (A[i] == target) {

return i;

}

}

return -1;

}

private void flip(int[] A, int index) {

int i = 0, j = index;

while (i < j) {

int temp = A[i];

A[i++] = A[j];

A[j--] = temp;

}

}

}